DOCUMENT RESUME

ED 348 981 IR 015 660

AUTHOR Tsai, Shu-Er
TITLE Development of Schema Knowledge in the Classroom: Effects upon Problem Representation and Problem Solution of Programming.
PUB DATE 92
NOTE 57p.; Paper presented at the Annual Conference of the American Educational Association (San Francisco, CA, April 20-24, 1992).
PUB TYPE Reports - Research/Technical (143) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC03 Plus Postage.
DESCRIPTORS *Aptitude Treatment Interaction; Computer Science Education; High Schools; *Instructional Effectiveness; *Intermode Differences; *Problem Solving; *Programing; Qualitative Research; Schemata (Cognition); Statistical Analysis
IDENTIFIERS BASIC Programing Language; *Problem Representation

ABSTRACT

Students with a semester or more of instruction often display remarkable naivety about the language that they have been studying and often prove unable to manage simple programming problems. The main purpose of this study was to create a set of problem-plan-program types for the BASIC programming language to help high school students build plans for unstructured BASIC programs with a schema induction intervention. With an experimental versus control group design and taking individual differences into analyses, this study was also aimed at investigating different functions of aptitudes and instructional treatments on computer programming learning. Treatment efficacy was assessed in terms of problem representation and problem solution of programming in card sorting tasks and performance tasks. Subjects were students from four high school computer programming classes. The results showed that inductive reasoning ability and treatment effect both contributed significantly to problem representation and problem solution of programming. The experimental group did better in card sorting tasks and performance tasks than the control group. The schema induction treatment proved to be effective. This study provides a learning framework for classes in computer programming. Results of the data analyses are displayed in 11 tables and 3 figures and a sample problem set is appended. (Contains 43 references.) (Author/BBM)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☐ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

ED348981

Development of Schema Knowledge in the Classroom: Effects upon Problem Representation and Problem Solution of Programming

Shu-Er Tsai

Stanford University

Paper presented at the annual meeting of the American Educational Research Association at San Francisco, August 1992.

IR015660

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

Shu-Er Tsai

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

BEST COPY AVAILABLE

ABSTRACT

Students with a semester or more of instruction often display remarkable naivety about the language that they have been studying and often prove unable to manage simple programming problems. The main purpose of this study was to create a set of problem-plan-program types for the BASIC programming language to help high school students build plans for unstructured BASIC programs with a schema induction intervention. With an experimental versus control group design and taking individual differences into analyses, this study was also aimed at investigating different functions of aptitudes and instructional treatments on computer programming learning. Treatment efficacy was assessed in terms of problem representation and problem solution of programming in card sorting tasks and performance tasks. Subjects were students from four high school computer programming classes. The result showed that inductive reasoning ability and treatment effect both contributed significantly to problem representation and problem solution of programming. The experimental group did better in card sorting tasks and performance tasks than the control group. The schema induction treatment proved to be effective. This study provides a learning framework for classes in computer programming.

Considerable research has shown that most students at the elementary and high school levels do not attain an adequate level of competency from a semester or more of instruction in LOGO or BASIC (Kurland, Pea, & Mawby, 1986; Linn, 1985; Pea & Kurland, 1984a, 1984b). Students with a semester or more of instruction often display remarkable naivety about the language that they have been studying and often prove unable to manage simple programming problems (Perkins, Schwartz, & Simmons, 1988). The main purpose of this study was to create a set of problem-plan-program types for BASIC to help high school students build plans of the unstructured BASIC programming with a schema induction intervention. The potential effects of the intervention for students were better problem representation (i.e., deeper understanding of the problem) and better problem solving skills. In order to evaluate the effectiveness of the intervention, individual differences in aptitudes were taken into account. Thus, this study was also designed to investigate different effects of aptitudes and treatments on problem representation and problem solution in an introductory programming course.

BACKGROUND

At the high school level, most students learn programming piecemeal by looking at various programming language features. Most high school teachers tend to teach programming language features first, then assign a problem for students to work on by themselves. This "self-discovery" approach has not been found effective (Fay & Mayer 1988; Mayer, 1989; Nickerson, Perkins, and Smith, 1985; Perkins, Schwartz, and Simmons, 1988). For example, Nickerson et al. (1985) revealed that "students often produced programs without really comprehending how the programs worked" (p.277). This "disappointing reality", using Mayer's term (1988, p. 3) in discussing computer programming education, gave rise to a strong claim by most educational researchers that hands-on discovery approaches should be complemented with direct instruction or at least some structured guidance by a teacher.

One way to teach programming for students needing a more structured form of instruction is to use schemata or plans of programming. Schemata or plans of programming have been suggested recently by several educational and computer science researchers (Linn, 1985; Mayer, 1988; Soloway, 1985, 1986). Schemata or plans are generic program fragments that represent stereotypical action sequences in programming to accomplish a particular task or subtask. For example, a summing schema involves initializing a total variable, reading in input data, accumulating the total variable, and printing out the total. Many researchers use an expert model of programming to teach novices structured languages like PASCAL (Linn & Clancy, 1989; Soloway, 1986) and LISP (Soloway, 1985). Expert models of programming rely on schemata or plans as a central notion. Tutoring systems like Johnson and Soloway's PROUST for PASCAL (1983) and Anderson's LISP tutor (1984,1989) also use the notion of schemata or plans.

But having the notion of schemata or plans is not enough. There needs to be some method to teach schemata effectively. Researchers (Gick & Holyoak, 1983; Gentner, 1983; Vosniadou & Ortony, 1989) found that schemata could be easily learned by mapping underlying structures (i.e., solution structures) between concrete examples (i.e., analogs). In their studies, students exposed to analogs with surface similarity and deep similarity were able to induce a schema by a process of mapping -- finding a set of one-to-one correspondences between aspects of one analog and aspects of another. Then, the induced schema (mapped identities) afforded the basis for analogical reasoning and problem solving. Based on analogical reasoning theory, schemata can be taught and learned effectively with the above schema induction approach. Schemata help students comprehend programming problems deeply, which in turn help them solve problems efficiently. Greeno & Simon (1988) claim that "patterns of information in the problem have to be recognized to determine that a problem solving operator can be applied" (p. 606). Expert problem solvers represent problems immediately in terms of core programming structures, which allow them to find elegant solutions. Novice problem solvers fixate on surface features of problems without comprehending the structure which can be translated into a program (McKeithen, Reitman, Rueter, Hietle, 1981). Patterns of information in the problem are possible to be "recognized" only if

problem solvers have learned the patterns or stored the patterns in their heads. For example, Chi, Feltovich, and Glaser (1981) have studied the behavior of experts and novices in representing physics problems. They asked subjects to sort physics problems on the basis of similarity in how they should be solved. They found that novices sorted the problems on the basis of surface features, that is, on the basis of similarity in the objects, physics terms, and physical configurations contained in the problems; whereas experts sorted the problems on the basis of deep structure, that is, on the basis of abstract, solution-relevant aspects of the problems. The current study claims that the schema induction approach to teaching programming improves students' understanding of programming problems. That is, if asked to sort the problems on the basis of similarity in how they should be solved, students with schema induction learning will sort the problems based on similarity of schemata instead of similarity of surface features. In addition, this study claims that the schema induction approach to teaching programming improves students' performance in solving programming problems. The argument is that with a repertoire of schemata in mind, students have a set of flexible and powerful techniques that allow them to solve many problems without inventing new code. In other words, schemata play an important role in problem representation that leads to correct problem solving. Students' problem representation and problem solving in programming would be significantly improved by teaching schemata of programming versus using a self-discovery approach. In order to evaluate the effectiveness of the schema induction approach, the learning outcomes of the schema induction group will be compared to that of the self-discovery group.

In evaluating the effectiveness of the schema induction approach, certain individual differences of learners need to be taken into account. Ideally, an instructional treatment is aimed at improving the learning of all participants in the situation. However, individual differences in aptitudes will influence the degree of achievement in a particular situation. That is, learners with different aptitude profiles learn better in different instructional situations (Cronbach & Snow, 1977). It is important to know what aptitudes are involved in the learning process in the proposed schema induction teaching method, in contrast to alternative methods, before judging its effectiveness.

In computer programming, students' cognitive aptitudes might influence learning differently in different treatments. For example, inductive reasoning ability has been shown to influence programming achievements (Snow, 1980; Ricardo, 1983; Langstaff, 1989). Research indicated that experts and novices represent problems and solve problems differently (Adelson, 1981, 1984; McKeithen, Reitman, Rueter, & Hirtle, 1981). Thus, prior programming experience and knowledge influence learning, but may do so differently depending on methods of instruction. With respect to conative aptitudes, persons differ in the tendency to engage in and enjoy effortful cognitive activity versus the tendency to minimize mental effort in processing incoming information. The degree of students' mindfulness as a tendency influences their learning (Salomon & Globerson, 1987). Since students show individual differences in cognitive aptitudes (inductive reasoning ability and prior programming experience) and conative aptitudes (mindfulness) in computer learning, it is important to take these aptitudes into account. Differential effects of aptitudes on computer learning need to be studied especially in relation to the effectiveness of the schema induction approach versus the self-discovery approach.

An aptitude process theory of learning from instruction (Snow, 1980; Snow & Lohman, 1984) provides a context for exploring the relationship of individual differences in aptitudes and computer programming learning for different instructional methods. The major claim of aptitude theory is that aptitudes often appear to interact with instructional methods. That is, aptitudes relate differently to learning outcomes under different instructional treatments. For example, direct and explicitly structured instruction benefits novices, young children, and students of low-ability, but is not necessarily highly appropriate for all learners (Doyle, 1983). Indirect and less structured instruction may provide able students with practice in applying their higher-order reasoning skills and in building their own comprehension of the material. Also, according to this theory, "aptitude process differences exist before, operate through, but are also produced by, instruction to account for individual differences in learning outcomes" (Snow, 1980, p. 29). That is, individual differences in aptitudes change in a dynamic functional system during the process of instruction to account for individual differences in learning outcomes. For example, students of low-ability in a

schema induction condition might find direct and structured instruction conducive to solving problems, and thus feel confident in themselves and willing to put effort into the task. They might progress to more able functioning as structure is removed. In contrast, students of low-ability in a self-discovery, unstructured instructional condition may feel chaotic and helpless, and thus not self-efficacious for the task given. They might regress to even less able performance, or to denial and escape. On the other hand, students of high-ability may not differ much in structured and unstructured conditions from the beginning to the end of the instruction, since able students may elaborate their own structures and discover necessary schematic knowledge to solve problems by themselves. However, it is possible that highly structured instruction may cause an "interference" effect for able students and decrease their interest in learning. Thus students' aptitudes may interact with the schema induction approach versus the self-discovery approach differently in influencing learning outcomes.

Research Questions

The motivation of this study was to improve classroom learning of programming in application of research on schema induction and analogical reasoning (Genter, 1989, Gick & Holyoak, 1983, Vosniadou & Ortony, 1989) as well as research on aptitude for learning from instruction (Snow, 1980; Snow & Lohman, 1984). This study addressed the following questions: (1) To what extent does schema induction affect student problem representation and problem solution of programming? (2) Is problem representation of programming highly related to problem solution of programming? (3) How do aptitudes interact with instructional treatments to affect student learning in computer programming?

METHOD

Subjects

Subjects were 75 students from four BASIC high school computer programming classes at two high schools. Two classes (one from each school) were taught as the experimental group with

the schema induction intervention; the other two classes (one from each school) were taught as the control group with a self-discovery approach. All the students used the same textbook, worked on the same worksheets and assignments, and took the same tests. However, the experimental group was taught using a different instructional method from the control group.

Measures

Pretest: Aptitude Measures

Raven's Advanced Progressive Matrices Set 2 (Raven, 1977): This test was designed to assess inductive reasoning ability or fluid ability (Gf). The examinee was required to solve problems presented in abstract figures and designs.

Mindfulness-as-a-Tendency Questionnaire (Salomon & Globerson, 1987): This questionnaire consists of 32 items to measure one's tendency to be more or less mindful in general. A typical item is "I like to carefully plan things ahead of time, thinking of the many possibilities that might come up." Subjects were asked to choose one from fully agree to totally disagree on a 5-point scale.

Review Test of BASIC Programming Statements: This test was used to measure students' knowledge of BASIC programming statements before the experiment. Subjects were asked to select one procedure specification for each statement among three choices.

Card Sorting Problem Sets: A set of 20 introductory computer programming problems (see Appendix A) was constructed in which surface features (e.g. money, numbers, letters, and geometry) were crossed with secondary features involved in solution schemata (e.g. flag, restriction, menu, counting, and summing schema). Some problems were involved with multiple schemata. For example, the following four problems were categorized as menu schema problems: The first was related to letter problems; the second to geometry problems; the third to number problems; and the last to money problems.

> i. Write a program which allows the user to select different greetings in different languages, e.g. (1) print "Hello!" in English, or (2) print "Bonjour" in French, or (3) print "Aloha" in Hawaiian.

ii. Write a program that allows the user to choose to print the area of square given the length of any side, or to print the area of a circle given any diameter, or to quit the program.

iii.Write a program which allows the user to (1) take the square root of an inputted number, or (2) square the inputted number, or (3) cube the inputted number.

iv. Write a program where the user can choose to (1) calculate the cost of x hot dogs when the user types in x and each hot dog costs $.75, (2) calculate the cost of n large Pepsis where the user types in n and each Pepsi costs $.90, or (3) quit the program.

Subjects were asked to sort the computer programming problems into groups based on similarities of solution. They were given 25 minutes for the sorting aspect of the task and 10 minutes to complete the answer sheet. They were then asked to label each problem group and also give a reason for choosing the label.

Think Aloud Protocol: Ten subjects of the control groups (5 each group) and 14 subjects of the experimental groups (7 each group) were given training in the skill of thinking aloud and then asked to think aloud, while writing program code on paper. Each think aloud procedure was video taped and the audio track was transcribed. This task was used to measure problem solving skills in programming before intervention.

Posttest: Outcome Measures

Card Sorting Problem Sets: The same as the pretest.

Programming Performance Test: All subjects were asked to write a program on a computer. A printout of the program was coded. This test was used to measure all subjects' problem solving skills in programming (see Appendiix B).

Think Aloud Protocol: The same student sample was used in the posttest as in the pretest (Two students in the control groups were not able to participate in the posttest). Students were asked to think aloud, while writing another program code on paper. Each think aloud procedure was video taped and the audio track was transcribed.

Design

Students in both classes in both schools were taught the basics of BASIC programming statements by their teachers for one month. They were taught the syntax of the BASIC

programming language before the experimental manipulation. Then, the experimental group was given the schema induction intervention; the control group was given regular classroom instruction with a self-discovery approach (described later). Both groups in both schools were taught by the researcher for three weeks during the experimental period. This study involved an experimental versus control group design with repeated measures as described above.

Procedure

Before the experimental manipulation was undertaken, all students were administered the Raven's Advanced Progressive Matrices Set 2, the Review Test of BASIC Programming Statements, and the Card Sorting Problem Sets. Twenty four students performed think aloud protocols in programming. Students were paid $10 to participate in the think aloud task after school.

Instructional material

Students in the control group were given the same assignments as the experimental group. They had more time to work on the assignments individually and did not have explicit, guided instruction on program structures. This approach is called a self-discovery approach.

Before working on each assignment, the experimental group was taught explicitly about the structures of 5 schemata (counting, summing, stop-value, menu, and bullet-proofing schema), which had been developed by the programming research group at Harvard School of Education (see Perkins et al., 1988) and re-organized by the author of this study. All schemata were introduced through the same schema induction procedure outlined in Figure 1 and described afterwards.

Insert Figure 1 about here

Procedure for Schema Induction Intervention: One example

As may be seen in Figure 1, the schema induction intervention involves 6 steps:

STEP 1: Students were given two analog problems to find out the similarities and differences between the problems. The following was an example:

A. Write a program to create a program that will record a bank transaction. The user should have the option of choosing one of three actions: deposit, withdrawal, or quit. If the user selects deposit, he/she inputs the amount of money to be added to the account. If the user selects withdrawal, he/she inputs the amount of money to be withdrawn. In both cases, the balance appears on the screen after the transaction has been completed. Then the program ends. Assume the user's starting balance is $100.00.

B. Write a program to test the user. Choice A is that a goldfish is a mammal, and choice B is that a goldfish is a fish. Tell him "yes" or "no" based on his answer and also tell him the correct answer. He can also choose to quit the program without answering the question.

Both problems A and B allow the user to select among a number of choices. Both problems are similar in the solution structure, although dissimilar in some surface features. Students are encouraged to write down what they find.

STEP 2: They were given flowcharts for two analogs and encouraged to compare them. The notions of flowcharts were introduced first. Then two flowcharts were given. The flowcharts were very similar, although one dealt with money; another dealt with letters.

STEP 3: Students were encouraged to write down the code for the flowcharts. At this time, they were given concrete program examples of mapping the structures to the code.

STEP 4: Students were then given the abstract structure between the examples on an overhead projector with verbal explanations and diagram presentations from the instructor. A chunk of program statements to achieve the goal of the schema was emphasized and introduced explicitly to the whole group.

STEP 5: A similar programming problem matched with the same schema was given as an assignment. Students were asked to finish the task within two class periods.

STEP 6: At the end of learning each schema, students received feedback on their assignments and obtained a detailed example of code with a full diagram of the whole program. Students were encouraged to map the example code to their own code.

After the experimental manipulation was undertaken, all students sorted the card sorting problem sets the same as in the pretest. All students also wrote a program on a computer for the programming performance task. Twenty two students the same as in the pretest performed think aloud protocols in programming and were paid another $10 for their participation after school.

RESULTS

Two schools from two very different school districts participated in this study. The two schools were different in terms of student characteristics, teaching and learning environments, and school locations. Since this study did not investigate school context effects on students' learning, only important student aptitudes such as reasoning ability, previous experience and mindfulness were taken into consideration in the data analysis. Ideally, a student would be randomly assigned to the experimental class and would have a matched replicate (with regard to initial aptitude levels) in the control class. However, it was impossible for this study to do so, because students could not easily change their class schedules to fit random assignment.

Since random assignment of students into the experimental and control groups was not possible, classroom effects and school effects needed to be controlled. In order to do this, a preliminary data analysis was conducted such as the T-test of initial aptitudes between the control group and the experimental group, and between school A and school B. The results did not show any significant differences between the experimental and the control groups in terms of students' reasoning ability, mindfulness, BASIC programming syntactical knowledge and schematic knowledge. The two instructional groups were comparable. However, there were significant differences between the two schools in students' reasoning ability and BASIC programming syntactical knowledge. In order to further analyze school effects, the intercorrelations among

aptitude measures for both schools jointly and for each school individually were calculated. The resutls indicated that the structures of the correlations of aptitude measures for School A and School B individually and for the two schools jointly were pretty similar. It was then decided to pool two schools together in analyzing the data and test the ATI hypotheses for the experimental groups (one from school A and the other from school B) and the control groups (one from school A and the other from school B). However, in order to adjust the school effects, school would be treated as a dummy variable in the testing hypotheses. School x treatment interaction effect was also analyzed.

Problem Representation

Student problem representation was assessed on pretest and posttest card sorting tasks. Quantitative and qualitative analyses were conducted on pretest and posttest data separately to see if any change of problem representation for each group had taken place after the intervention.

Quantitative Analysis: Latent Partition Analysis

Latent partition analysis (LPA) was formulated by Wiley (1967) to analyze the categorizations of problems created by subjects. A major structural hypothesis of LPA is that a latent partition underlies the manifest partitions; that is, it is assumed that each problem belongs to a latent category and that the manifest categories are derived by dividing and combining the latent categories. The model also assumes that in sorting the problems, the subjects may not follow the latent categories exactly. There are probabilities, constant within categories and across persons, that errors will occur in the partitioning process. Therefore, latent and manifest groupings might not be identical. The purpose of LPA is to detect the common, underlying category structure which was assumed to have given rise to the diverse, fallible partitions made by the subjects.

The raw data for LPA was a sequence of different partitions of the same set of items. Four joint occurrence matrices (S) were created separately for the pretests and posttests of both the experimental group and the control group, indicating the proportion of subjects who placed each pair of elements in the same cluster. Eigenvalues were computed for the joint occurrence matrices (S) of 20 card sorting problems. These eigenvalues helped determine the number of latent

categories in the solution for each group. This study used as many categories as there were eigenvalues greater than one. Once the number of latent categories was determined, the Phi matrix was derived as explained by Wiley (1967). The Phi matrix indicated the partitioning of the items (in this case, 20 programming problems) into latent categories. This line of analysis was performed to determine the degree to which subjects in each group agreed that certain problems belong to the same cluster. In other words, the results showed the latent partition of problems (in this case, the knowledge structure of programming problems) on the pretest and posttest for each group.

Qualitative Analysis of Categorization Labels

One advantage of the card sorting tasks in this study was to obtain a name for each cluster (latent category) by looking at each category name labelled by the subjects. For example, if Problem A, B, and C are in one latent category based on LPA, the names for the category with at least Problem A, B, and C in it were listed. In order to investigate how subjects in each group labeled the categories, the author consolidated the categories of problem statements generated by the subjects into 5 or 6 clusters based on the results of LPA. A cluster name was chosen according to the same name given by the majority of the subjects in each group. In addition, for each label, the rater judged it as either surface-feature oriented or solution-feature oriented. For example, Problem F, J, and 5 were in one cluster. A "money" label for the cluster with at least Problem F, J, and 5 in it was judged as surface feature oriented, because all the three problems in the same cluster were involved with money. Another example, Problem K, X, and 5 were in one cluster. An "accumulator" or "summing up" label for it was judged as solution oriented, because it implied some function of the program. A cluster was judged as a whole as either surface-feature oriented or solution oriented based on the labels given by the majority of the subjects in each group. A detailed coding scheme for the categorization labels is available on request. The author and another independent coder developed the coding schemes for the qualitative analysis together, and then coded data separately. The interrater reliabilities of the pretest and posttest card sorting tasks were

.89 and .93 respectively. This line of analysis was conducted to see how subjects categorize the problems according to their explanations of classification.

The following are the results of the above described analyses:

Knowledge Structure and Its Categorization Description of the Experimental Group

Pretest: Table 1 shows 6 clusters of pretest card sorting items for the experimental group from LPA and indicates the labels given by the experimental group on the pretest under 6 clusters. 37% of the students put at least Problem K, X, and 5 together and gave it a surface feature oriented "total" label. The other surface feature oriented labels are "money", "figures", "grouping", "number", and "print". Totally, 66% used the above surface oriented labels to categorize Problem K, X, and 5. Only 32% of students used solution oriented labels such as "math", "number addition", "accumulator", "calculate", "counter", and "sum up". 2% was residual. It appears that the "total" cluster is a surface feature oriented category.

35% of the students gave cluster 2, which includes problems A, F, N, R, Z, and 3, a solution oriented label "choice". The other similar labels are "menu" and "option". Totally, 41% gave the above solution oriented labels. However, 56% gave surface oriented labels, including "letters", "weird", "ask question", "input", "print", "variables", "password", "problem", and "reader." This cluster is a surface feature oriented category.

The third cluster is also a surface feature oriented category, labelled as "negative numbers" by 25% of students in this group. The other surface feature oriented labels were "money", "check", '"coordinates", "dimensions", "read", and "series". Totally, 68% of students used the above surface feature oriented labels versus 31% gave solution oriented labels such as "input terminate", "flag", "scanning", and "sorters". Another surface feature oriented category is labelled by 33% of students as "area/shape". Similar labels were "geometry", "figures", "square", and 'dimension", adding up to 70% totally. The other 29% gave solution oriented labels such as "choice", "calculator", "formula", "math", and "specific parameter".

The "numbers" category obviously is a surface oriented one, labelled by 37% of this group. Totally, 68% of students used surface oriented labels such as "ten numbers",

"integer", '"relation", and "write program", in addition to the "numbers" label. The other 31% used solution oriented labels such as "calculation", "if...then", "number addition", "only certain one", and "restriction". 50% of students label the last category as "acceptable" and "read" -- surface feature oriented names. The other 50% labelled it as "restriction", "exclusives", and "only certain one" solution oriented names.

Insert Table 1 about here

It appeared that experimental group categorized problems mainly based on surface features before the intervention. Categorization according to surface features is a typical novice behavior as reported by Chi, Feltovich, and Glaser (1981) and other researchers in a wide range of fields.

Posttest: The posttest LPA results for the experimental group were very different from those of the pretest, yielding only 5 clusters (Table 2). In terms of categorization labels, Table 2 indicates that more than 60% (from 62% to 73%) of the experimental group used solution-oriented labels to categorize the five clusters. For cluster 1, which includes Problem A, F, O, P, and 3, totally 70% used solution oriented labels such as "menu", "choice", "option", "pick one" to name the category. For cluster 2, which includes Problem K, X, 5, and 6, 73% used solution oriented labels such as "accumulator", "math", and "calculator" to name the category. Of those 73%, 61% used "accumulator" to indicate the function of the problems in this category. 73% used solution oriented labels including "restriction", "corrector", "if then", "bullet proofing", "filter", and "limiter" to name the third cluster, which includes Problem B, R, W, Y; 62% used solution oriented labels such as "terminator", "read data", "only certain one", and "stop value" to name the fourth cluster, and 68% used "counter", "accumulator", "keep tracking", and "tally", all solution oriented labels, to name the fifth cluster.

Insert Table 2 about here

The experimental group was able to sort the cards based on goal features of the problems after the schema induction intervention, in contrast to the same group's performance during the pretest. It meant that the schema induction approach changed subjects' knowledge structure in a significant way.

Knowledge Structure and Its Categorization Description of the Control Group

Pretest: Table 3 shows 6 clusters of pretest card sorting items for the control group. It appeared that this group categorized problems mainly based on surface features as the experimental group did on the pretest. In regard to surface-feature oriented labels, 56% used either "total", "money", "check", "drawer", "figure out", or "problem" to name the first category; 69% used either "words", "printer", "allow user", or "or" to name the second cluster; 83% used either "area/shape", "geometry", "square", "figure", "length", "multiply", or "print" to name the fourth cluster; 81% used either "accept", "money", or "select" to name the fifth cluster; and 73% used either "numbers", or "integer" to name the sixth cluster. Of those in the 73%, 55% used "numbers" to name it.

Insert Table 3 about here

Posttest: Unlike the experimental group, the posttest LPA results for the control group was not much different from those of the pretest (see Table 4). Four of the 6 clusters were labelled based on surface features of the problems as follows: 64% used either "area/shape", "geometry", "square", or "figure" to name the second cluster; 84% used either "checking" or "input" to name the third cluster, 92% used either "numbers", "negative", "catch", "integer", "never ending", "selected output", or "values" to name the fourth category; and 80% used either "letters", "printing", "accept to print", "ask the user", "input", "money", "printers", "question", or "1234 quit" to the sixth cluster. However, 72% used solution oriented labels such as "addition", "math", "calculation", "computing numbers", and "low processing" to the first cluster; 82% used either "specific input", "certain numbers", "combination checking", "exact answer", "rejection", or

"restriction" solution oriented labels to name the fifth cluster. Subjects in the control group were not able to sort the cards completely based on solution features of the problems. They relied much on the surface features of the problems. Even after they worked on assignments with similar solution-oriented structures as the card sorting problems, they still sorted the cards based on surface-oriented features.

Insert Table 4 about here

In conclusion, after analyzing the pretest and posttest card sorting tasks for the experimental group and the control group, the experimental group did significantly better on posttest card sorting tasks than on the pretest, whereas the control group did not show significant improvement. The knowledge structure of the experimental group improved significantly from pretest to posttest (i.e., from focus on surface features to solution features). In contrast, the knowledge structure of the control group reflected more surface features of the problems than solution features of the problems on both the pretest and posttest card sorting tasks. The schema induction intervention improved the experimental group's problem representation.

Problem Solution

In order to get more detailed information on how subjects solve complex programming problems, some of the subjects were asked to think aloud while they were writing programs. The think aloud protocols of sample students were analyzed based on whether students showed schema structures and plan-oriented problem-solving procedure (i.e., program design) while writing a program. A schema was divided into three parts: the initialization (I), the execution (E) or process, and output (O). More formally, a schema was defined as { I [E] O}, where [] denotes zero or more repetitions. The output part is usually implicit in planning,so it's not used in protocol analysis. In terms of schema structures: If all essential parts within a schema appeared, the program was coded as "complete"; if either the I part or the E part did not appear, the program was coded as "incomplete"; if both the I part and the E part did not appear, the program was coded as

"no structure". In terms of program design within each schema (i.e., intra-schema): When an emergence order of EI appeared, it was coded as "bottom up" ; when an emergence order of IE appeared, it was coded as "top down". In terms of program design between two schemata (i.e., inter-schema): If two schemata were developed simultaneously, it was coded as "breadth-first"; if one schema was developed after another, it was coded as "depth-first". Two raters scored the think aloud protocols independently. The interrater reliability was 8.9 for the pretest and .94 for the posttest. The reliabilities were high because of good instructions for coding. A detailed coding scheme for the think aloud protocols is available on request.

Table 5 shows the results of chi-square analysis of schema structures in think aloud protocols between the experimental group and the control group. There was no difference between the two groups on the pretest problem in which only the menu schema was involved. However, on the posttest problem, there were significant differences between the two groups on menu schema ($\chi^2 = 6.01$, $p < .05$) and terminator schema ($\chi^2 = 6.08$, $p < .05$) in favor of the experimental group. More students in the experimental group showed complete restriction schema structures than the control group, although the difference was not statistically significant.

Insert Table 5 about here

In terms of plan-oriented problem solving procedures, two lines of analyses were conducted. Table 6 shows the results of chi-square analysis of intra-schema processes (i.e., the development within the same schema) in think aloud protocols between the experimental group and the control group. Table 7 shows the results of chi-square analysis of inter-schema processes (i.e., the development between two schemata) in think aloud protocols between the experimental group and the control group. There was a tendency for both the experimental group and the control group to display more top-down program design within a schema than bottom-up design, if the schema structures existed in their protocols. However, the experimental group showed more breadth-oriented program design between two schema processes than the control group by initializing both

schemata at almost the same time, then executing the two schemata simultaneously. Both schemata were embedded. In contrast, more students in the control group displayed depth-first oriented program development. They tended to finish one schema first, then work on the second schema. Two schemata were done separately, one by one. There was a significant difference between the experimental group and the control group in developing the program using the menu schema and restriction schema ($\chi^2 = 5.46$, $p < .05$).

Insert Table 6 & 7 about here

To sum up, this line of analysis indicated that if schema structure appeared in students' think aloud protocols, students tended to demonstrate top-down (forward) and breadth-first programming development. After the intervention, the experimental group displayed more complete schema structures (e.g., menu schema and terminator schema) than the control group. Thus, the experimental group demonstrated more top-down and breadth-first programming development than bottom-up and depth-first development. In contrast, the control group demonstrated more depth-first programming development than breadth-first development, although they demonstrated more top-down than bottom-up development as the experimental group did.

Correlation of Problem Representation and Problem Solution

The previous analysis demonstrated how programming knowledge structures and problem solving procedures differed between the experimental group and the control group and how they changed from pretest to posttest within each group. For each student, the scores for both learning outcomes were also computed: One is for problem representation (i.e., card sorting task) and the other is for problem solution (i.e., programming performance task).

Quadratic Assignment Paradigm Analysis of Card Sorting Task

Quadratic assignment paradigm (QAP) was developed by Hubert and Schultz (1976) as a general approach to data analysis tasks characterized by the use of proximity matrices. It can be used to compare the actual partitioning of N objects (e.g., stimulus elements) to an a priori

partitioning. That is, QAP contrasts the degree of correspondence between an empirically obtained categorical structure and an a priori target categorical structure with the degree of correspondence between two such structures that would be expected to occur by random chance. In this study, it was used to compare the categorical structure imposed by each student on the programming problems with that of a predefined schematic structure. The resulting statistics were z scores for each individual. Higher z values indicated a greater match to the target organization.

Analysis of Programming Performance Task

All subjects were given a programming performance task to work on the computer. Their printouts were collected and scored. There were two schemata involved in the performance task: accumulation schema and restriction schema. Therefore, the scoring of the performance task was divided into two parts. Within each schema, certain features (i.e., program code) were specified as essential to accomplish the goal of the program. For each specified feature, students who did not write any relevant program code received no points; those who tried to write relevant code but made some mistakes received 1 point; those who wrote relevant code without any mistakes received 2 points. The total score for each subject was created by adding the points across 8 important features (4 for each schema). Two raters scored the performance test independently. The interrater reliability was .95. Each subject was given an average score based on the judgment of the two raters. A higher score means the student showed better programming problem solving skills than a student with a lower score.

Problem representation and problem solving were significantly correlated for the whole sample ($r=.45$, $p< .01$). It indicated that deep understanding of programming problem features was positively correlated with programming performance as in the domains of physics and geometry.

Aptitude Treatment Interaction on Learning Outcomes

In order to evaluate the effectiveness of the intervention, individual differences in cognitive and conative aptitudes were taken into account to investigate different effects of aptitudes and

treatments on learning outcomes. Before testing the ATI hypotheses, descriptive statistics for aptitudes and outcomes are discussed.

Scores from the review test (i.e., syntactical knowledge) and the QAP results of card sorting tasks (i.e, schematic knowledge) were standardized in the total sample and combined to form a composite labeled Gc. Scores from the Raven Progressive Matrices test, also standardized in the total sample, formed Gf. The Gc and Gf were combined to form two orthogonal indices in order to investigate their combined and differential importance: SUM, the sum of the Gc and Gf, represented general ability, and DIFF, Gc minus Gf, represented the ability profile difference. Positive values of DIFF thus indicate students with a higher Gc compared to their Gf, and conversely negative values indicate a larger Gf compared to the earned Gc. This procedure has advantages over using Gc and Gf directly in the analysis (Cronbach and Snow, 1977). First, hypotheses can be ordered so that the general-ability hypothesis can be tested independently from the more exploratory differential-ability hypothesis. Second, SUM and DIFF are uncorrelated, whereas, Gc and Gf usually correlate (r=.53 in this study). Thus, using SUM and DIFF provides a less ambiguous interpretation than using Gc and Gf directly.

Means and standard deviations of all outcomes for the experimental and the control groups are reported in Table 8. On the average, students in the experimental group obtained higher scores on the two outcome measures -- problem representation and problem solution. Table 9 gives correlations between outcome and aptitude measures. Significant correlations between SUM and problem representation and problem solution appeared in both treatment and control groups, while correlations for DIFF were not significant in either group. However, the correlations of DIFF and problem solution showed differences between two treatments (-.23 for the experimental group and .12 for the control group), although they were not statistically significant. Correlations between mindfulness and problem representation and problem solution were generally lower than those between SUM and outcomes. However, there were small differences between the two treatments in the correlations of mindfulness and the two outcome measures.

Insert Table 8 & 9 about here

Research on ATI assumes that aptitudes often appear to relate differently to learning outcome under different instructional treatments (Cronbach & Snow, 1977). In this study, the different correlations of DIFF and problem solution between the experimental group and the control group implied ATI. Table 10 shows that the main effects of school, SUM, and treatment on problem representation were significant. None of the interaction effects was significant. This line of analysis explained 42% of the variance on problem representation. Even without school effect in the analysis, the second model accounted for 40% of the variance. The results indicated that students' general ability and instructional treatments contributed significantly to their problem representation. On the one hand, students with higher general ability did better on card sorting task than those of lower general ability. That is, students with higher abilities had better understanding of the problems. On the other hand, students with the schema induction intervention learned more on problem representation of programming than the control group.

Insert Table 10 about here

Table 10 also shows that the main effects of school, SUM, and treatment were significant on problem solution. Significant school x treatment and DIFF x treatment interaction effects were found ($p < .1$). This line of analysis explained 44% of the variance on problem solution. Without school effect in the analysis, the second model accounted for 41% of the variance and the DIFF x treatment interaction effect was also significant ($p < .05$).

In order to further analyze the school x treatment effect on problem solution, the ATI hypothesis was tested for School A and School B separately as shown in Table 11 Panel 2. For School A, the main effects of SUM and treatment effects were significant. The other effects were not significant. The regression model accounted for 53% of the variance. For school B, the main

effect of SUM was significant. Treatment effect was not significant. However, DIFF x treatment and Mindful x treatment interaction effects were significant ($p < .05$). The model accounted for 62% of the variance. Schools interacted with treatments differently.

Insert Table 11 about here

Indication of ATI

To understand how DIFF interacted with instructional treatments for the whole sample, regression analysis of DIFF on problem solution was conducted for the experimental group and the control group separately . The unstandardized regression coefficients within each treatment were plotted graphically for interpretation in Figure 2. The relative advantage of high differential ability (Gc minus Gf) students was most pronounced in the control group as shown in the first panel of Figure 2. In other words, students with higher Gc compared to Gf did better in the control group. Students with lower Gc compared to Gf did better in the experimental group. The schema induction appeared particularly helpful for low differential ability students, who had lower Gc than Gf.

Insert Figure 2 about here

Similarly, for school B, regression analysis of DIFF on problem solution was conducted for the experimental group and the control group separately. The third panel of Figure 2 shows that the regression slope for the experimental group intersected with the regression slope for the control group. It indicated that students of lower differential ability (Gc minus Gf) did better in the experimental group than in the control group, whereas those of higher differential ability did better in the control group than in the experimental group. As hypothesized, more experienced students (high Gc students) did better in the self-discovery approach, whereas less experienced students did

better in the schema induction intervention. For School A, no DIFF x treatment interaction effect was found.

Figure 3 Panel 3 shows the results of regression slopes of mindfulness on problem solution for the experimental group and the control group in School B. As may be seen in Figure 3 Panel 3, the regression slope for the experimental group intersected with the regression slope for the control group. As hypothesized, higher mindfulness students did better in the control group than in the experimental group, whereas low mindfulness students did better in the experimental group than in the control group. The schema induction appeared particularly helpful for low mindfulness students in School B. No interaction effect on problem solution was found in School A.

Insert Figure 3 about here

In summary, the ATI hypothesis was supported in the regression analyses of problem solution outcome. As a whole, there was a tendency for high differential ability students to do better in the control group than in the experimental group. Especially for School B, there was a tendency for high mindfulness students to do well in the control group and for low mindfulness students to do well in the experimental group. However, for School A, no interaction effect on problem solution was found.

DISCUSSION

Considerable research has shown that most students with a semester or more of instruction often display remarkable naivety about the language that they have been studying and often prove unable to manage simple programming problems. This study attempted to help students deeply understand the structures of programming and successfully solve programming problems with a schema induction intervention that can be implemented in regular classroom teaching/learning settings.

Most research on teaching or learning has searched for teaching acts such as structuring and questioning that are correlated with student achievement, on the assumption that there are certain ways of teaching that will help all students. The literature on Aptitude-Treatment Interactions (ATI), however, suggests that the effects of teaching/learning depend on student individual differences (Cronbach & Snow, 1977). Some students perform better with a certain instructional treatment, while other students do better with a different treatment. Assuming that high aptitude students learn better under less directed and less structured instruction and low aptitude students learn better under more directed and more structured instruction, this study investigated interactions between learner characteristics and teaching methods. It tried to answer the question: How can we teach students with different aptitudes effectively based on their needs in computer programming learning?

From another perspective, this study was different from previous research on problem solving. Previous research on problem representation and problem solution laid out the differences between experts and novices (e.g. Chi, Feltovich, & Glaser, 1981; Larkin, 1983, in physics; Adelson, 1981, Rist, R. S., 1989, in computer programming). This study went beyond the differentiation of experts and novices. It aimed at helping novices learn problem solving patterns and problem solving skills, which experts already use. This study demonstrated an effective way to accomplish this based on Gick and Holyoak's (1983) theoretical study on schema induction and Gentner's theoretical (1983) study on structure-mapping. It demonstrated one way of applying psychological mechanisms to computer programming teaching and learning. Computer science education can be integrated with psychological theories to benefit each other one way or another (Anderson, Conrad, & Corbett, 1989; Lehrer, Randle, & Sancillio, 1989).

The schema induction approach to teaching and learning of computer programming was proved to be effective. It not only helped students develop schematic knowledge structure in problem representation, but also facilitated expert-like problem solving skills in problem solution. The following discusses the results:

Problem Representation

The schema induction intervention changed students' problem representation from focusing on surface features of problems (pretest phase) to focusing on solution features (goal structures) of problems (posttest phase). Researchers on analogical reasoning claimed that mapping an analog to a schema would be simpler than mapping one analog with another, because in the former case, it is only necessary to map identities, rather than both identities and differences (Gick & Holyoak, 1983). This study was able to test and support this claim. In this study, the experimental group learned a schema through some guidance and worked on an assignment related to the schema afterwards. The control group didn't learn the schema but worked on the same assignment independently. When doing card sorting tasks, students in the experimental group were able to sort the cards into categories with the schema structures as taught in the class. Presumably, they mapped 20 card sorting problems to the schemata (mapped identities) they have learned. In contrast, the control group sorted the cards into surface-feature oriented categories. They searched for the identities and differences on the basis of physical configurations contained in the problems. It was likely that they mapped each problem with the other problems by looking at differences and identities at the same time. Students were able to transfer what they learned in the schema induction intervention to problems with similar schema structures.

Problem Solution

The schema induction intervention improved students' knowledge structure as well as problem solving procedures in programming performance (i.e., problem solution). Students with the intervention tended to have complete schemata and displayed top-down and breadth-first-oriented development of programming. This programming behavior was similar to that of programming experts. This continued Jeffries, Turner, Polson, and Atwood's (1981) research on programming showing that experts generated a solution by breadth-first development, while novices generated a solution by depth-first development. Depth-first design by novices were also found by Anderson and others (Anderson, Farrell, & Sauers, 1984; Pirolli, 1986).

This study supported the claim that the efficiency and accuracy of the solver's subsequent thinking in problem solving should be determined by the quality, completeness, and coherence of the initial representation of the problems (Greeno & Simons, 1988). That is, students' initial problem representation could predict their subsequent problem solving performance in programming. It replicated the findings in other domains such as physics (Chi, Feltovich, & Glaser, 1981) and geometry (Greeno, 1983). This study addressed how students' knowledge structure changed from novice-like problem representation to expert-like problem representation. It also demonstrated that with schema knowledge in mind, students were able to display top-down, forward and breadth-first program development.

The ATI Effects

This study is in part an investigation of how aptitudes interact with instructional structures on learning outcomes. However, a sample size of 75 was inadequate for the assessment of the interactions. According to Cronbach and Snow, 100 students per treatment are advisable to reduce Type-II errors and obtain confident conclusions. "The ATI effect is a difference in regression slopes. The sampling error of a difference in correlations is 1.4 times the sampling error of each correlation, and differences in slope are similarly hard to pin down" (Cronbach & Snow, 1977, p. 46). Consequently, it was unlikely to obtain a satisfactory test of the ATI in this study because of the small sample size.

As a matter of fact, significant differential ability (Gc minus Gf) and treatment interaction effects were found in the regression analysis of programming performance tasks, especially for School B in this study. The schema induction intervention did not benefit students with more prior experience compared to their reasoning ability as much as it benefitted students with less prior experience relative to their reasoning ability. It helped students of less prior experience relative to their reasoning ability by giving them component assemblies they would not have produced on their own. Also, for School B, significant mindfulness and treatment interaction effects were found in the regression analysis of programming performance tasks. Low mindfulness students did better in the schema induction intervention, while high mindfulness students did better in the

self-discovery approach. The schema induction intervention was very helpful for low mindfulness students, because given specific and continuous guidance, they were forced to induce effort expenditure in mindful abstraction and thus improve their learning. However, high mindfulness students reacted when given specific guidance, because they lost opportunities to invest mental effort by themselves.

This study predicted that low aptitude students would benefit more from the schema induction approach; high aptitude students would benefit more from the self-discovery approach. The different impact of Gc and Gf as measured in this study, did enter differently into outcome on aptitude relations. The impact of mindfulness also entered differently into outcome on aptitude relations. Although the impact of general ability did not enter significantly differently into outcome on aptitude relations, the trend did suggest that students with low general ability learned better from the schema induction approach whereas students with high general ability learned better from the self-discovery approach. This study supported previous ATI research finding that aptitudes interact with type and medium of instructional treatments (Snow, 1977; Yalow, 1980). Thus no one treatment seems right for all students.

The School Effects

One of the major findings of this study was that aptitude and treatment interacted differently in the two schools. In School A, treatment effects on problem representation and problem solution. There were no interaction effects on various outcomes. In School B, treatment effect on problem representation was statistically significant. There was no significant treatment on problem solution. However, there were significant DIFF x treatment and mindfulness x treatment interaction effects on problem solution outcome.

The above complicated results were inconclusive. One reason for such findings was that different sample sizes in two schools yielded different significant results. This is one reason why ATI researchers (Cronbach & Snow, 1977) recommend researchers to look for result patterns or trends instead of significance levels. There was a trend for School B to have ATI effects on problem solution learning outcome, but not for School A. Different ATI patterns for the two

BEST COPY AVAILABLE

schools in this study should be examined further to understand how school context influenced student learning, which was not the focus of this study. School A students had higher reasoning ability and syntactical knowledge of BASIC than School B students. Also 77% of students owned a personal computer at home in School A; only 10% of students owned a personal computer at home in School B. In general, School A students seemed to come from families of higher social economic status than School B students. It was possible that the two schools from two different school districts produced very different school contexts for students to learn (McLaughlin, M., Talbert, J., Bascia, N., 1991).

In a broader sense of learning, person x situation interactions do exist in different school environments and in different classroom settings. The existence of interactions in this study seems to support Cronbach's (1975) assertion that human teaching-learning processes are more complex than the simple generalizations that psychologists have often constructed to describe them. Those who continue in the "main-effect mentality" are bound to fail in their attempts to explain human behavior.

CONCLUSIONS AND IMPLICATIONS

This study supported the hypothesis that schema-based instructional design in a domain can provide the basis for better instruction in that domain. This study also provided evidence that neither aptitude nor instructional treatment alone can fully describe learning outcome. Interactions between aptitudes and treatments do exist and can be demonstrated. Researchers who seek the "best" teaching method should be asking the question, "Best for whom?" and "Best for what?" Similarly, teachers should be aware that there is no one best way of teaching, but rather, different teaching approaches are likely to be best for different kinds of students.

The implications for educational practice are clear. Adaptive teaching in programming is necessary. Higher-aptitude learners are more ready or have a better inner environment, using Snow's term (1980), to learn from instruction than lower-aptitude learners. However, according

to aptitude process theory (Snow & Lohman, 1984), outer instructional environments can be chosen or designed to capitalize on strengths and compensate for weakness in the inner environments of different students. The schema induction intervention was designed to facilitate mindful abstraction, presumably by giving lower-aptitude learners or novices component assemblies they can not produce on their own. It benefitted students in the experimental group, especially low-aptitude learners. It carried a person to a better inner environment, which may open the door to many careers that would otherwise remain closed to students without a programming background. In regular classroom settings, teachers can manage the class in small groups with mixed ability students and encourage students to ask questions and explain to each other. Meanwhile, teachers could give the schema induction intervention to those students, who need more help, based on previous experience and ability.

Additional research is necessary to confirm the findings of this study. First, further research is needed to address how schema induction can facilitate the learning of different ability students under different school environments. That is, school effects on teaching and learning need to be studied further, in relation to individual differences. Second, more research is needed to explain ATI in teaching and learning and it should be supplemented with more basic research in information processing. Improved methods of distinguishing Gc from Gf are needed to explore their combined impact and differential impact on learning. These methods may emerge as we gain a better understanding of these abilities through further research.

Finally, researchers should examine both immediate and delayed outcomes of a cognitively-based instruction design. The present study examined only the immediate outcomes. Directed research on delayed outcomes may probe more deeply into the relationships among instructional material and aptitudes, if we are truly interested in the long-term impact of instruction.

In conclusion, the findings of this study have both psychological and educational importance. From a psychological point of view, the study provides evidence that the search for generalizations about human behavior must take into account the interaction of variables in the human environment. From an educational point of view, it provides information on what types of

students do better with what teaching approaches. Such information will be useful to educators, teachers, administrators, and parents, who desire to promote student achievement and improve student attitude.

REFERENCES

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. Memory and Cognition, 2, 422-433.

Adelson, B. (1984). When novices surpass experts: The difficulty of a task may increase with expertise. Journal of Experimental Psychology: Learning, Memory, and Cognition, 10 (3), 483-495.

Anderson, J. R., Farrell, R., & Sauers, R. (1984). Learning to program in LISP. Cognitive Science, 8, 87-129.

Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill acquisition and the LISP tutor. Cognitive Science, 13, 467-505.

Atwood, M. E., & Ramsey, H. R. (1978). Cognitive structure in the comprehension and memory of omputer program: An investigation of computer program debugging. Army Research Institute.

Chi, M. T. H., Feltovich, P. J., & Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. Cognitive Science, 5, 121-152.

Chi, M. T. H., Glaser, R., & Reese, E. (1982). Expertise in problem solving. In R. Sternberg (Ed.), Advances in the psychology of human intelligence, vol. 1. Hillsdale, NJ: Lawrence Erlbaum Associates.

Cronbach, L. J. (1975). Beyond the two disciplines of scientific psychology. American Psychologist, 30, 116-127.

Cronbach, L. J., & Snow, R.E. (1977). Aptitude and instructional methods: A handbook for research on interactions. New York: Irvington Publishers, Inc.

Doyle, W. (1983). Academic work. Review of Educational Research, 53, 159-199.

Fay, A. L., & Mayer, R. E. (1988). Learning LOGO: A cognitive analysis. In R. E. Mayer (Ed.) Teaching and learning computer programming. (pp.55-74). Hillsdale, NJ: Lawrence Erlbaum Associates.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analog. Cognitive Science, 7, 155-170.

Gentner, D. (1989). The mechanisms of analogical learning. In S. Vosniadou & A. Ortony (Eds.), Similarity and analogical reasoning. New York, NY: Cambridge University Press.

Gick, M. L., & Holyoak, K. J. (1983). Schema induction and analogical transfer. Cognitive Psychology, 15, 1-38.

Greeno, J. G. (1983). Forms of understanding in mathematical problem solving. In S.G. Paris, G. M. Olson, & H. W. Stevenson (Eds.), Learning and motivation in the classroom. Hillsdale, NJ: Erlbaum.

BEST COPY AVAILABLE

Greeno, J. G., & Simon, H. A. (1988). Problem solving and reasoning. In R. C. Atkinson, R. J. Herrnstein, G. Lindzey, & R. D. Luce (Eds.), Stevens' Handbook of Experimental Psychology. (2nd edition). Vol. 2: Learning and Cognition. New York: John Wiley & Sons.

Johnson, W. L., & Soloway, E. (1983). PROUST: Knowledge-based program understanding. Technical Report 295, Department of Computer Science, Yale University.

Kurland, D. M., Pea, R. D., Clement, C., & Mawby, R. (1986). A study of the development of programming ability and thinking skills in high school students. New York: Bank Street College of Education, Center for Children and Technology.

Langstaff, J. J. (1989). Problem representation and achievement in computer programming: The differential effects of inductive reasoning skills and computer programming experience. Unpublished doctoral dissertation, University of Iowa.

Larkin, J. H. (1983). Problem representation in physics. In D. Gentner & A. L. Stevens (Eds.), Mental models. Hillsdale, NJ: Erlbaum.

Lehrer, R., Randle, L., & Sancilio (1989). Learning preproof geometry with LOGO. Cognition and Instruction, 6, 159-184.

Linn, M. C. (1985). The cognitive consequences of programming instruction in classrooms. Educational Researcher, 14, 14-29.

Linn, M. C., & Dalbey, J. (1985). Cognitive consequences of programming instruction: Instruction, acess, and ability. Educational Psychologist, 20, 191-206.

Linn, M. C., & Clancy, M. J. (1989). The case for case studies of programming problems. Paper presented at the meeting of the American Educational Research Association, San Francisco, CA.

Mayer, R. E. (1988). Introduction to research on teaching and learning computer programming. In R. E. Mayer (Ed.) Teaching and learning computer programming. (pp.1-12). Hillsdale, NJ: Lawrence Erlbaum Associates.

McKeithen, K. B., Reitman, J. S., Rueter, H. H., & Hirtle, S. C. (1981). Knowledge organization and skill differences in computer programmers. Cognitive Psychology, 13, 307-325.

McLaughlin, M. W., Talbert, J. E., Bascis, N. (Eds.) (1990). The contexts of teaching in secondary schools: Teachers' realities. New York, NY: Teachers College Press.

Nickerson, R. S., Perkins, D. N., & Smith, E. E. (1985). The teaching of thinking. Hillsdale, NJ: Lawrence Erlbaum Associates.

Pea, R. D., & Kurland, D. M. (1983). On the cognitive prerequisites of learning computer programming. Technical Report No.18. New York: Bank Street College of Education.

Pea, R. D., & Kurland (1984b). LOGO programming and the development of planning skills. (Rep. No. 16). New York: Bank Street College.

Perkins, D. N., Schwartz, S., & Simmons, R.(1988). Instructional strategies for the problems of novice programmers. In R. E. Mayer (Ed.) Teaching and learning computer programming. (pp.153-178). Hillsdale, NJ: Lawrence Erlbaum Associates.

Pirolli, P. L. (1986). A cognitive model and computer tutor for programming recursion. Human-Computer Interaction, 2, 319-355.

Ricardo, C. M. (1983). Identifying student entering characteristics desirable for a first course in computer programming. Unpublished doctoral disseration. Columbia University, Graduate School of Arts and Science.

Rist, R. S. (1989). Schema Creation in Programming. Cognitive Science, 13, 389-414.

Salomon, G., & Globerson, T. (1987). Skill may not be enough: The role of mindfulness in learning and transfer. International Journal of Educational Research, 7, 623-637.

Snow, R. E. (1977). Research on aptitude for learning: A progress report. In L. S. Shulman (Ed.), Review of research in education (Vol. 4). Itasca, IL: Peacock.

Snow, R. E. (1980). Aptitude Processes. In R. E. Snow, P. A. Federico, & W. E. Montague (Eds.) Aptitude, Learning, and Instruction : Cognitive Process Analyses of Aptitude (vol.1). Hillsdale, NJ: Lawrence Erlbaum Associates.

Snow, R. E., & Lohman, D. F. (1984). Toward a theory of cognitive aptitude for learning from instruction. Journal of Educational Psychology, 76, 347-376.

Soloway, E. (1985). From problems to programs via plans: The content and structure of knowledge for introductory LISP programming. Journal of Educational Computing Research, 12, 157-171.

Soloway, E. (1986). Learning to program = learning to construct mechanisms and explanations. Communications of the ACM, 29, 850-858.

Vosniadou, S., & Ortony, A. (1989) (Eds.). Similarity and analogical reasoning. New York, NY: Cambridge University Press.

Wiley, D. E. (1967). Latent partition analysis. Psychometrika, 32, 183-193.

Yalow, E. S. (1980). Individual differences in learning from verbal and figural material. Unpublished doctoral disseration. Stanford University, School of Education.

Table 1

Qualitative Description of Latent Category Names of Pretest Card Sorting for Experimental Group (N=38)

Cluster 1 (K,X,5) N=24		Cluster 2 (A,F,N,Z,3) N=31		Cluster 3 (D,J,T) N=16		Cluster 4 (O,P,4) N=24		Cluster 5 (B,M,Y,6) N=19		Cluster 6 (R,W) N=10	
Name	%	Name	%	Name	%	Name	%	Name	%	Name	%
Total	37	*Choice*	35	Negative number	25	Area/ shape	33	Numbers	37	Acceptable	40
Money	13	Letters	13	*Input terminate*	13	Geometry	17	Ten numbers	16	*Restriction*	30
Math	08	Weird	10	Money	13	*Choice*	13	*Calculation*	11	*Exclusive*	10
Number addition	08	Ask question	06	Check	06	Figures	08	Integer	05	*Only certain one*	10
Accumulator	04	Input	06	Coordinates	06	Square	08	*If...then*	05	Read	10
Calculate	04	Print	06	Dimensions	06	*Calculator*	04	*Number addition*	05		
Counter	04	Variables	06	*Flag*	06	Dimension	04	*Only certain one*	05		
Figures	04	*Menu*	03	Read	06	*Formula*	04	Relation	05		
Grouping	04	*Option*	03	*Scanning*	06	*Math*	04	*Restrictor*	05		
Number	04	Password	03	Series	06	*Specific parameter*	04	Write program	05		
Print	04	Problem	03	*Sorters*	06						
Sum up	04	Reader	03								
Surface[a]	66		56		68		70		68		50
Solution[b]	32		41		31		29		31		50

Note. Cluster members for each cluster are listed, using card labels, in parentheses under each cluster. The name lists within each cluster are sorted by descending proportions. The first name for each cluster is used as the cluster name. The italicized names are solution oriented labels. The non-italicized names are surface-oriented labels.

a Surface-feature oriented labels.

b Solution-feature oriented labels.

Table 2

Qualitative Description of Latent Category Names of Posttest Card Sorting for Experimental Group (N=38)

Cluster 1 (A,F,O,P,3,4) N=38		Cluster 2 (K,X,5,6) N=31		Cluster 3 (B,R,W,Y) N=30		Cluster 4 (D,J,T,J) N=24		Cluster 5 (M,N) N=16	
Name	%	Name	%	Name	%	Name	%	Name	%
Menu	34	*Accumulator*	61	*Restriction*	47	*Terminator*	46	*Counter*	50
Choice	26	Total	16	*Corrector*	10	-999	13	Checking	18
Print	11	*Math*	09	Accepting	07	Print	08	Checker	13
Allows	05	Area	03	*If...then*	07	*Read data*	08	*Accumu-lator*	06
Option	05	*Calculator*	03	Input	07	Choice	04	*Keep tracking*	06
Quit	05	Figure out	03	*Bullet proofing*	03	Investigator	04	*Tally*	06
Pick one	05	Money	03	*Filter*	03	Numbers	04		
For next	03			Five	03	*Only certain one*	04		
Geometry	03			*Limiter*	03	Series	04		
Input	03			Only	03	*Stop-value*	04		
				Pick one	03				
				Steps	03				
Surface[a]	30		25		26		37		31
Solution[b]	70		73		73		62		68

Note. Cluster members for each cluster are listed, using card labels, in parentheses under each cluster. The name lists within each cluster are sorted by descending proportions. The first name for each cluster is used as the cluster name. The italicized names are solution oriented labels. The non-italicized names are surface-feature oriented labels.

a Surface-feature oriented labels.

b Solution-feature oriented labels.

Table 3

Qualitative Description of Latent Category Names of Pretest Card Sorting for Control Group (N=35)

Cluster 1 (K,5) N=22		Cluster 2 (A,3) N=23		Cluster 3 (D,J,M,N,T,Z) N=17		Cluster 4 (O,P,X,4) N=25		Cluster 5 (F,R,W) N=11		Cluster 6 (B,Y,6) N=11	
Name	%	Name	%	Name	%	Name	%	Name	%	Name	%
Total	27	Words	43	Checking	24	Area/ shape	28	Accept	36	Numbers	55
Math	14	*Choice*	17	Negative numbers	12	Geometry	24	Money	27	Integer	18
Average addition	09	Printer	13	Programs	12	Square	12	Select	18	*Average + addition*	09
Money	09	Allow user	09	Input	12	*Choice*	08	*Channel*	09	*Calculation*	09
Processing input	09	*If...then*	04	*Calcuation*	06	*Calculation*	04	*Limited*	09	*Math*	09
Calculator	05	Or	04	Catch	06	*Decision*	04				
Check	05	*Selection*	04	Chart	06	Figure	04				
Drawer	05	*You are boss*	04	*Counter*	06	*Formula*	04				
Figure out	05			Find	06	Length	04				
Problem	05			Judge	06	Multiply	04				
Sum	05			*Math*	06	Print	04				
Surface[a]	56		69		83		80		81		73
Solution[b]	42		29		17		20		18		23

Note. Cluster members for each cluster are listed, using card labels, in parentheses under each cluster. The name lists within each cluster are sorted by descending proportions. The first name for each cluster is used as the cluster name. The italicized names are solution oriented labels. The non-italicized names are surface-feature oriented labels.

a Surface-feature oriented labels.

b Solution-feature oriented labels.

Table 4

Qualitative Description of Latent Category Names of Posttest Card Sorting for Control Group (N=35)

Cluster 1 (F,K,5,6) N=24		Cluster 2 (O,P,X,4) N=17		Cluster 3 (M,N,Z) N=13		Cluster 4 (D,J,T) N=13		Cluster 5 (B,R,W,Y) N=12		Cluster 6 (A,3) N=18	
Name	%	Name	%	Name	%	Name	%	Name	%	Name	%
Addition	21	Area/ shape	35	Checking	76	Numbers	37	*Specific input*	42	Letters	33
Math	16	*Choose*	29	*Accumu-lator*	08	Negative	15	Accept points	16	*Choice*	12
Calcu-lation	16	Geometry	17	*Counter*	08	Catch	08	*Certain numbers*	08	Printing	12
Computing numbers	15	Square	06	Input	08	Integer	08	*Combination checking*	08	Accept to Print	05
Total	08	Figure	06			Never ending	08	*Exact answer*	08	Ask the user	05
Money problems	08	*Formula*	06			Selected output	08	*Rejection*	08	Input	05
Series	04					*Terminator*	08	*Restriction*	08	Money	05
Figure out	04					Values	08			Printers	05
Collected input	04									*Processing*	05
Low processing	04									Question	05
										1234 quit	05
Surface[a]	28		64		84		92		16		80
Solution[b]	72		35		16		8		82		17

Note. Cluster members for each cluster are listed, using card labels, in parentheses under each cluster. The name lists within each cluster are sorted by descending proportions. The first name for each cluster is used as the cluster name. The italicized names are solution oriented labels. The non-italicized names are surface-feature oriented labels.

a Surface-feature oriented labels.

b Solution-feature oriented labels.

Table 5

Chi-Square Analysis of Schema Structures in Think Aloud Protocols between Experimental and Control Group

		Response (%)				
		No structure	Incomplete	Complete	χ^2	df[c]
Pretest						
Menu schema	CO[a]	40.00	30.00	30.00		
	EX[b]	28.57	42.86	28.57		
					.49	2
Posttest						
Menu schema	CO	12.50	50.00	37.50		
	EX	7.14	7.14	85.71		
					6.01*	2
Restriction schema	CO	25.00	12.50	62.50		
	EX	14.29	0.00	85.71		
					1.56	1
Terminator schema	CO	37.50	0.00	62.50		
	EX	0.00	0.00	100.00		
					6.08*	1

[a] CO = Control group (Pretest, N=10; posttest, N=8).

[b] EX = Experimental group (Pretest N=14; posttest N=14).

[c] When df=1, a 2 x 2 chi square is reported, obtained by pooling the "No structure" and "incomplete" categories. This was done where expected frequencies for some cells in the original 2 x 3 table were too small.

* $P < .05$.

Table 6

Chi-Square Analysis of Intra-Schema Processes in Think Aloud Protocols between Experimental and Control Group

		Response (%)				
		Top-down	Bottom-up	Not applicable	χ^2	df[c]
Pretest						
Menu schema	CO[a]	50.00	10.00	40.00		
	EX[b]	71.43	0.00	28.57		
					2.37	1
Posttest						
Menu schema	CO	87.50	0.00	12.50		
	EX	92.86	0.00	7.14		
					.43	1
Restriction schema	CO	62.50	12.50	25.00		
	EX	85.71	0.00	14.29		
					2.12	1
Terminator schema	CO	62.50	0.00	37.50		
	EX	100.00	0.00	0.00		
					6.08*	1

[a] CO = Control group (Pretest, N=10; posttest, N=8).
[b] EX = Experimental group (Pretest N=14; posttest N=14).
[c] When df=1, a 2 x 2 chi square is reported, obtained by pooling the "Bottom up" and "Not applicable" categories. This was done where expected frequencies for some cells in the original 2 x 3 table were too small.

* $P < .05$.

Table 7

Chi-Square Analysis of Inter-Schema Processes in Think Aloud Protocols between Experimental and Control Group

		Response (%)				
		Breadth-first	Depth-first	Not applicable	χ^2	df[c]
Posttest						
Menu schema & Restriction schema	CO[a]	37.50	37.50	25.00		
	EX[b]	85.71	0.00	14.29		
					5.46*	1
Restriction schema & Terminator schema	CO	25.00	37.50	37.50		
	EX	57.14	28.57	14.29		
					2.49	2
Terminator schema & Menu schema	CO	25.00	37.50	37.50		
	EX	64.29	28.57	7.14		
					4.28	2

[a] CO = Control group (Pretest, N=10; posttest, N=8).
[b] EX = Experimental group (Pretest N=14; posttest N=14).
[c] When df=1, a 2 x 2 chi square is reported, obtained by pooling the "Depth-first" and "Not applicable" categories. This was done where expected frequencies for some cells in the original 2 x 3 table were too small.

* $P < .05$.

Table 8

Means and Standard Deviations of Outcome Measures

	Total Sample		School A		School B	
	CO[a] (N=36)	EX[b] (N=39)	CO (N=20)	EX (N=25)	CO (N=16)	EX (N=14)
Problem Representation	2.01 (2.68)	4.29 (2.96)	2.45 (2.97)	4.69 (3.26)	1.48 (2.28)	3.60 (2.32)
Problem Solution	8.18 (4.06)	11.18 (3.78)	8.53 (3.62)	12.06 (3.51)	7.62 (4.83)	9.27 (3.82)

Note. Standard deviations are presented in parentheses.

[a] CO = Control group.

[b] EX = Experimental group.

Table 9

Correlations Between Aptitude and Outcome Variables

Aptitudes	Problem Representation Outcome: Experimental Group (N=39)	Control Group (N=36)	Total Sample (N=75)
SUM	.50*	.53*	.50*
DIFF	.18	.12	.15
Mindfulness	.10	.23	.11
Aptitudes	**Problem Solution Outcome: Experimental Group (N=39)**	**Control Group (N=36)**	**Total Sample (N=75)**
SUM	.46*	.55*	.49*
DIFF	-.23	.12	-.06
Mindfulness	.22	.29	.21

Table 10

Summary of Regression on Problem Representation and Problem Solution Outcome for Total Sample

	Problem Representation			Problem Solution		
	d.f.	Mean Squares	F value	d.f.	Mean Squares	F value
Full Model (1)	9	29.98		9	55.02	
School	1	21.02	3.38#	1	59.03	5.11*
SUM	1	142.14	22.85**	1	217.60	18.84**
DIFF	1	15.58	2.51	1	1.55	.13
Mindfulness	1	1.56	.25	1	.68	.06
Treatment	1	79.79	12.83**	1	118.66	10.08**
School x Treatmt	1	6.07	.98	1	40.14	3.48#
SUM x Treatmt	1	1.37	.22	1	15.73	1.36
DIFF x Treatmt	1	.21	.03	1	35.96	3.11#
Mindful x Treatmt	1	2.13	.34	1	5.82	.50
Residual	61	6.22		54	11.54	

R-Sqaure=.42 for problem representation.
R-Square=.44 for problem solution.

Full Model (2)	7	37.55		7	65.35	
SUM	1	161.74	26.37**	1	274.01	23.20**
DIFF	1	16.39	2.67	1	2.02	.17
Mindfulness	1	1.53	.25	1	.66	.06
Treatment	1	78.45	12.79**	1	120.28	10.18**
SUM x Treatmt	1	.10	.02	1	1.54	.13
DIFF x Treatmt	1	1.02	.17	1	51.91	4.40*
Mindful x Treatmt	1	3.63	.59	1	7.05	.60
Residual	63	6.13		56	11.80	

R-Sqaure=.40 for School problem representation.
R-Square=.41 for School problem solution.

Note.
SUM= Gc + Gf. DIFF= Gc - Gf.

# P < .10 * P < .05 ** P < .01

Table 11

Summary of Regression on Problem Representation and Problem Solution Outcomes for Each School

	School A			School B		
	d.f.	Mean Squares	F value	d.f.	Mean Squares	F value
Problem Representation						
Full Model	7	26.72		7	11.88	
SUM	1	87.74	11.17**	1	154.53	12.59**
DIFF	1	26.93	3.43#	1	.96	.22
Mindfulness	1	.10	.01	1	.23	.05
Treatment	1	71.24	9.07**	1	13.27	3.06#
SUM x Treatmt	1	.31	.04	1	5.09	1.18
DIFF x Treatmt	1	.52	.07	1	.65	.15
Mindful x Treatmt	1	.20	.03	1	8.42	1.94
Residual	34	7.85		21	4.33	
R-Sqaure=.41 for A R-Square=.48 for B						
Problem Solution						
Full Model	7	48.24		7	37.30	
SUM	1	129.30	14.05**	1	90.46	8.63*
DIFF	1	14.45	1.57	1	13.03	1.24
Mindfulness	1	4.74	.52	1	15.67	1.50
Treatment	1	158.65	17.24**	1	.23	.02
SUM x Treatmt	1	6.83	.74	1	11.52	1.10
DIFF x Treatmt	1	5.13	.56	1	56.13	5.36*
Mindful x Treatmt	1	18.54	2.01	1	74.07	7.07*
Residual	33	9.20		15	10.48	
R-Sqaure=.53 for School A. R-Square=.62 for School B.						

# P < .10 * P < .05 ** P < .01

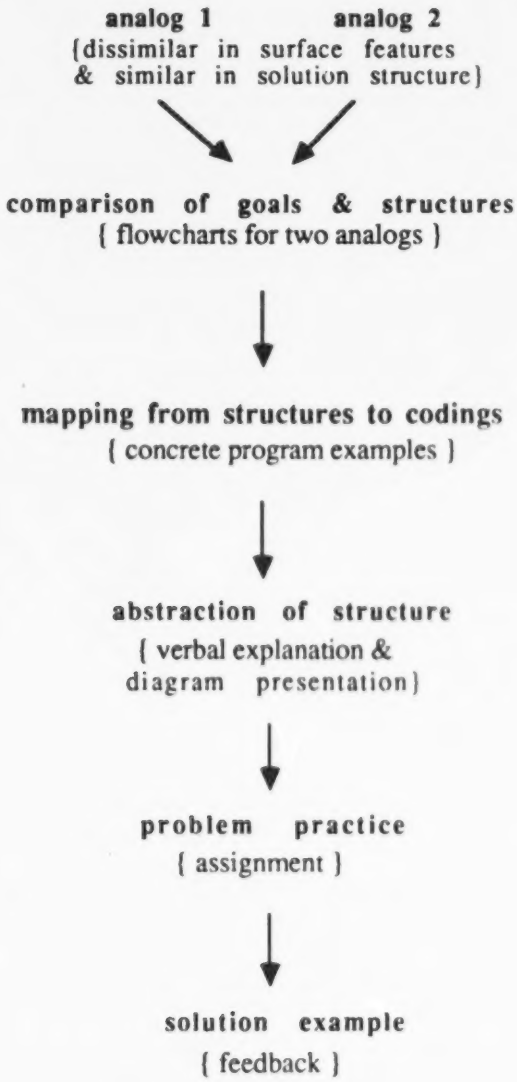
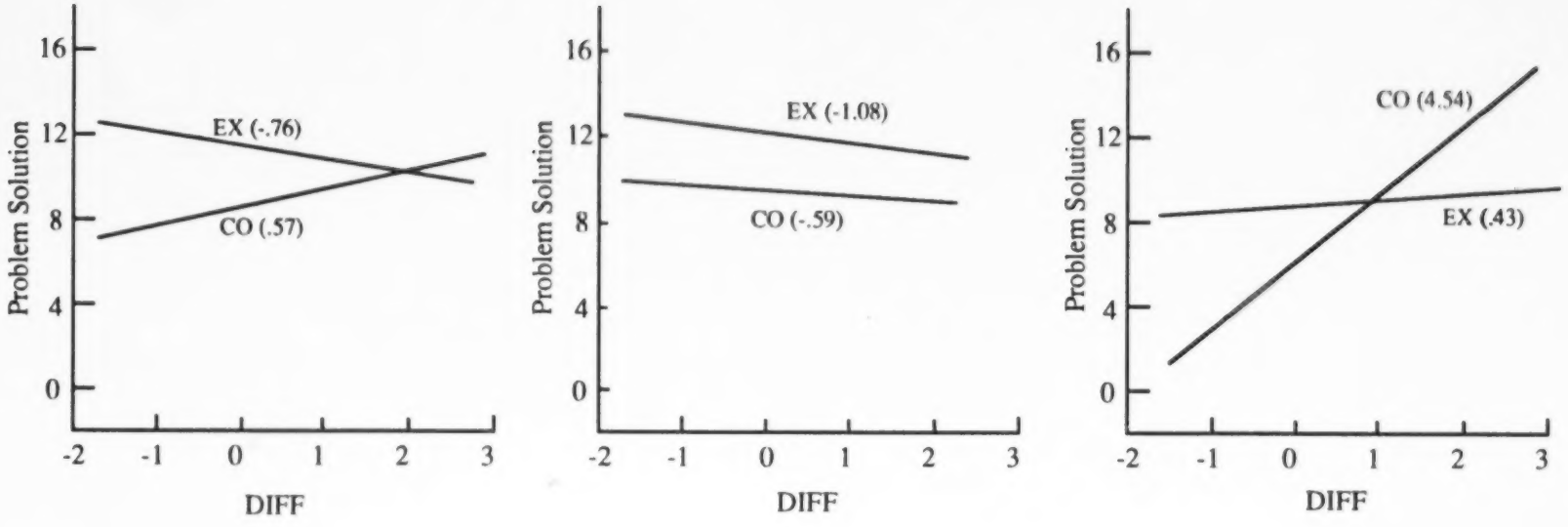


Figure 1 : Procedure for Schema Induction Intervention

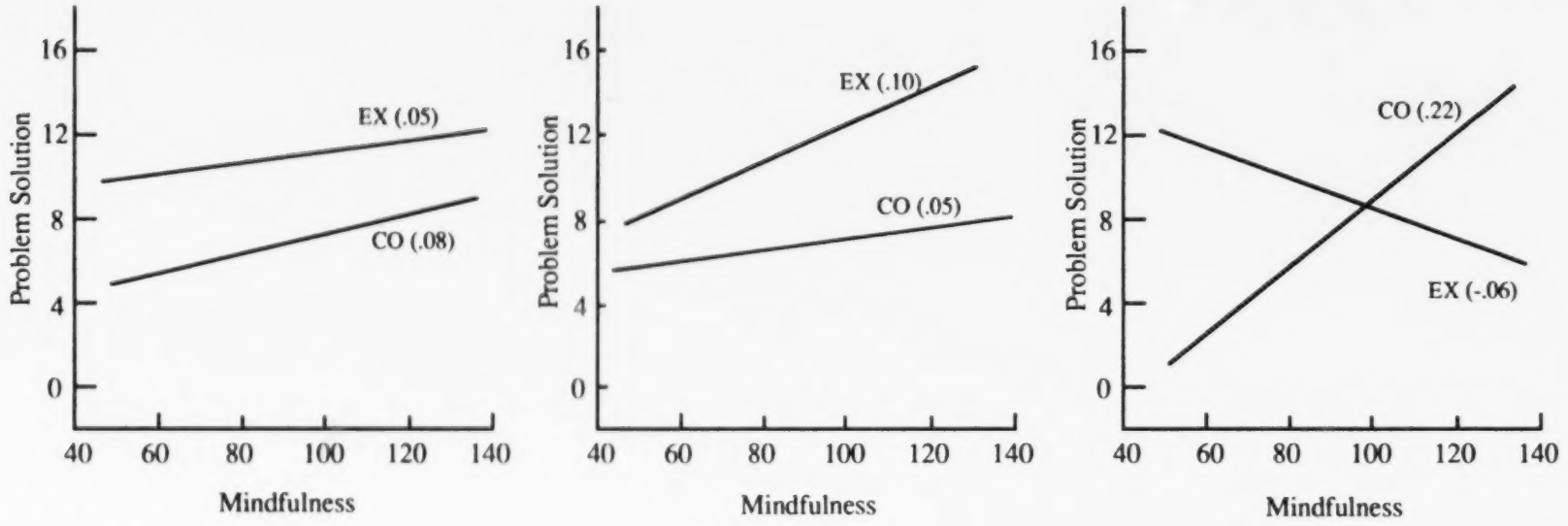


(1) For Whole Sample

(2) For School A

(3) For School B

Figure 2: Relation of Problem Solution to DIFF, with Unstandardized Regression Coefficients Shown in Parentheses for Experimental Group(EX) and Control Group (CO).



(1) For Whole Sample

(2) For School A

(3) For School B

Figure 3: Relation of Problem Solution to Mindfulness, with Unstandardized Regression Coefficients Shown in Parentheses for Experimental Group (EX) and Control Group (CO).

APPENDIX A

CARD SORTING PROBLEM SET

PART 1: Problems Organized by Surface Features

Money Related

J: Write a program to print out the range of parking tickets given out by a police officer in one day. User types in each ticket value and terminates sequence with a -99. Then write out the number of tickets between $0.00 and 10.00, inclusively, and then the number between $10.01 and $50.00, inclusively, and then $50.01 and $100.00, inclusively, and then over $100.00.

F: Write a program where the user can choose (1) Calculate the cost of x hot dogs when the user types in x and each hot dog costs $.75, or (2) calculate the cost of n large Pepsis where the user types in n and each Pepsi costs $.90, or (3) quit the program.

W: Write a program which accepts only acceptable money values like 2.50 or 45.00, which can be completely divided by 0.05 (a nickel). Entries such as 45.23 or 8.34 would be illegal because there would be non-integer numbers of nickels for that amount of money. Tell the user if he typed in a legal or illegal value.

K: Write a program to figure out the total amount of deposits for an account for one day. User types in each deposit and is asked if he has another deposit to enter. When there are no more deposits, print out the number of deposits.

5: Write a program to figure out the total income of a salesman for a year. The user types in the income for each month. The program prints out his yearly income.

Geometry

T: Write a program that accepts point coordinates (e.g. 1,2 or 4,3) and then finds the point closest to the origin (0,0), given that all points are different distances from the origin. Terminate the input of point coordinates by typing: -9999,-9999.

O: Write a program that will let the user choose to print the area of a square given the length of any side, or to print the area of circle given any diameter, or to quit the program.

4: Write a program which allows the user to print the area of a equilateral triangle, a square, or a regular hexagon. After the user chooses, program asks for the length of a side of the figure and prints the area. Other than the three kinds (3, 4, and 6 equal sides) of figures would not be accepted.

X: Write a program to add up the total square footage of a house. The user inputs the square footage for each room and the computer asks him if there are any more rooms. When all rooms have been entered, print out the total square footage of the house.

Numbers

D: Write a program to find the smallest number in a series of numbers. The user enters a series a numbers one by one. The number -999 means the user has finished entering his numbers. Print out the minimum of that series.

P: Write a program which allows the user to (1) take the square root of an inputted number, or (2) square the inputted number, or (3) cube the inputted number.

Y: Write a program that asks the user to input a one digit number (a number less than 10). If he inputs a one digit number, tell him that he typed in a correct response. Otherwise, tell him he didn't.

M: Write a program to check the number of even integers and odd integers in a series of 10 inputted integers.

6: Write a program to print out the sum of ten numbers. The user enters in the numbers and the program prints out the sum.

B: Write a program that allows the user to enter in 5 numbers and it prints out the average of those numbers. But there is a restriction that only numbers between 1 and 100 can be entered. If a number not in that range is entered, print that the number is unacceptable and re-ask for another number. At the end of the program, print the average.

Letters/Words

Z: Write a program that continuously asks the user for one letter on each input. Take only the first character, if more than one letter is typed in on each input (e.g. take J for John if John is typed in). A numerical input means the user has finished entering his letters. When the program is finished, it says if any of the inputted letters were 'Z'.

3: Write a program which allows the user to (1) print "Hello.", or (2) print "Bonjour" or (3) print "Aloha".

R: Write a program which accepts as input only the words "square", "triangle", "rhombus", "parallelogram", "trapezoid", "rectangle", and "quadrilateral". Any other words would not be accepted.

N: Write a program to check the number of "yes" answers to the following questions:
(1) Are you a high school student?
(2) Are you taking any computer classes?
(3) Do you have a computer at home?
(4) Do you want a job related to computers in the future?
(5) Do you think computers play a part in our modern life?

A: Write a program that asks the user if he wants to (1) print "Mary had a little lamb.", (2) print "Jack and Jill ran up the hill.", (3) print "Pop goes the weasel.", or (4) quit. If the user doesn't choose 1,2,3, or 4, tell the user of the error and reprint the menu. After the user chooses a correct response and the program performs appropriately, ask if the user wants to do this again.

PART 2: Problems Organized by Solution Features

Stop-Value (or Flag) Schema

J: Write a program to print out the range of parking tickets given out by a police officer in one day. User types in each ticket value and terminates sequence with a -99. Then write out the number of tickets between $0.00 and 10.00, inclusively, and then the number between $10.01 and $50.00, inclusively, and then $50.01 and $100.00, inclusively, and then over $100.00.

T: Write a program that accepts point coordinates (e.g. 1,2 or 4,3) and then finds the point closest to the origin (0,0), given that all points are different distances from the origin. Terminate the input of point coordinates by typing: -9999,-9999.

D: Write a program to find the smallest number in a series of numbers. The user enters a series a numbers one by one. The number -999 means the user has finished entering his numbers. Print out the minimum of that series.

Z: Write a program that continuously asks the user for one letter on each input. Take only the first character, if more than one letter is typed in on each input (e.g. take J for John if John is typed in). A numerical input means the user has finished entering his letters. When the program is finished, it says if any of the inputted letters were 'Z'.

Menu (or Choice) Schema

F: Write a program where the user can choose (1) Calculate the cost of x hot dogs when the user types in x and each hot dog costs $.75, or (2) calculate the cost of n large Pepsis where the user types in n and each Pepsi costs $.90, or (3) quit the program.

O: Write a program that will let the user choose to print the area of a square given the length of any side, or to print the area of circle given any diameter, or to quit the program.

P: Write a program which allows the user to (1) take the square root of an inputted number, or (2) square the inputted number, or (3) cube the inputted number.

3: Write a program which allows the user to (1) print "Hello.", or (2) print "Bonjour" or (3) print "Aloha".

4: Write a program which allows the user to print the area of a equilateral triangle, a square, or a regular hexagon. After the user chooses, program asks for the length of a side of the figure and prints the area. Other than the three kinds (3, 4, and 6 equal sides) of figures would not be accepted.

A: Write a program that asks the user if he wants to (1) print "Mary had a little lamb.", (2) print "Jack and Jill ran up the hill.", (3) print "Pop goes the weasel.", or (4) quit. If the user doesn't choose 1,2,3, or 4, tell the user of the error and reprint the menu. After the user chooses a correct response and the program performs appropriately, ask if the user wants to do this again.

Bullet-Proofing (or Checking) Schema

W: Write a program which accepts only acceptable money values like 2.50 or 45.00, which can be completely divided by 0.05 (a nickel). Entries such as 45.23 or 8.34 would be illegal because there would be non-integer numbers of nickels for that amount of money. Tell the user if he typed in a legal or illegal value.

4: Write a program which allows the user to print the area of a equilateral triangle, a square, or a regular hexagon. After the user chooses, program asks for the length of a side of the figure and prints the area. Other than the three kinds (3, 4, and 6 equal sides) of figures would not be accepted.

Y: Write a program that asks the user to input a one digit number (a number less than 10). If he inputs a one digit number, tell him that he typed in a correct response. Otherwise, tell him he didn't.

R: Write a program which accepts as input only the words "square", "triangle", "rhombus", "parallelogram", "trapezoid", "rectangle", and "quadrilateral". Any other words would not be accepted.

A: Write a program that asks the user if he wants to (1) print "Mary had a little lamb.", (2) print "Jack and Jill ran up the hill.", (3) print "Pop goes the weasel.", or (4) quit. If the user doesn't choose 1,2,3, or 4, tell the user of the error and reprint the menu. After the user chooses a correct response and the program performs appropriately, ask if the user wants to do this again.

B: Write a program that allows the user to enter in 5 numbers and it prints out the average of those numbers. But there is a restriction that only numbers between 1 and 100 can be entered. If a number not in that range is entered, print that the number is unacceptable and re-ask for another number. At the end of the program, print the average.

Summing Schema

5: Write a program to figure out the total income of a salesman for a year. The user types in the income for each month. The program prints out his yearly income.

X: Write a program to add up the total square footage of a house. The user inputs the square footage for each room and the computer asks him if there are any more rooms. When all rooms have been entered, print out the total square footage of the house.

6: Write a program to print out the sum of ten numbers. The user enters in the numbers and the program prints out the sum.

B: Write a program that allows the user to enter in 5 numbers and it prints out the average of those numbers. But there is a restriction that only numbers between 1 and 100 can be entered. If a number not in that range is entered, print that the number is unacceptable and re-ask for another number. At the end of the program, print the average.

Counting Schema

J: Write a program to print out the range of parking tickets given out by a police officer in one day. User types in each ticket value and terminates sequence with a -99. Then write out the number of tickets between $0.00 and 10.00, inclusively, and then the number between $10.01 and $50.00, inclusively, and then $50.01 and $100.00, inclusively, and then over $100.00.

K: Write a program to figure out the total amount of deposits for an account for one day. User types in each deposit and is asked if he has another deposit to enter. When there are no more deposits, print out the number of deposits.

M: Write a program to check the number of even integers and odd integers in a series of 10 inputted integers.

N: Write a program to check the number of "yes" answers to the following questions:
(1) Are you a high school student?
(2) Are you taking any computer classes?
(3) Do you have a computer at home?
(4) Do you want a job related to computers in the future?
(5) Do you think computers play a part in our modern life?

Z: Write a program that continuously asks the user for one letter on each input. Take only the first character, if more than one letter is typed in on each input (e.g. take J for John if John is typed in). A numerical input means the user has finished entering his letters. When the program is finished, it says if any of the inputted letters were 'Z'.

APPENDIX B

PROBLEMS FOR THINK ALOUD

Pretest:

Write a program where the user can choose (1) Calculate the cost of x hot dogs when the user types in x and each hot dog costs $.75, or (2) calculate the cost of n large Pepsis where the user types in n and each Pepsi costs $.90, or (3) quit the program.

Posttest:

Write a program that repeatedly asks the user if he wants to buy the following drinks: Pepsi ($.50 each), Coke ($.45 each), and 7up ($.48 each) and calculates the total cost of all drinks selected. The program asks the user what kind of drink and how many of each kind he wants to buy. If he types in other than the three kinds, tell him, "Sorry, not in store", and ask him to try again. Otherwise, the program asks for another selection. If the user doesn't want any more drinks, the program calculates the total cost of all drinks he has chosen and prints it out.

PROGRAMMING PERFORMANCE TEST

Write a program that allows the user to enter in 5 numbers and it prints out the average of those numbers. But there is a restriction that only numbers between 1 and 100 can be entered. If a number not in that range is entered, print that the number is unacceptable and re-ask for another number. At the end of the program, print the average.